Energy-Efficient Multiprocessor Scheduling for Flow Time and Makespan

Hongyang Sun[*] Yuxiong He[†] Wen-Jing Hsu[‡]

Abstract

We consider energy-efficient scheduling on multiprocessors, where the speed of each processor can be individually scaled, and a processor consumes power s^α if it runs at speed s, where $\alpha > 1$. A scheduling algorithm needs to decide both processor allocations and speeds for a set of parallel jobs whose parallelism can vary with time. The objective is to minimize the sum of overall energy consumption and some performance metric, which in this paper includes flow time and makespan. For both objectives, we present semi-clairvoyant algorithms that are aware of the instantaneous parallelism of the jobs but not their future information. We present U-CEQ algorithm for flow time plus energy, and show that it is $O(1)$-competitive. This is the first $O(1)$-competitive result for multiprocessor speed scaling on parallel jobs. We also consider, for the first time in the literature, makespan plus energy. We present P-FIRST algorithm and show that it is $O(\ln^{1-1/\alpha} P)$-competitive for parallel jobs consisting of fully-parallel and sequential phases, where P is the total number of processors. Moreover, we prove that P-FIRST is asymptotically optimal in this setting by providing a matching lower bound. In addition, we revisit non-clairvoyant scheduling for flow time plus energy, and show that N-EQUI algorithm is $O(\ln P)$-competitive. We then prove a lower bound of $\Omega(\ln^{1/\alpha} P)$ for any non-clairvoyant algorithm.

1 Introduction

Energy has been widely recognized as a key consideration in the design of modern high-performance computer systems. One popular approach to control energy is by dynamically changing the speeds of the processors, a technique known as *dynamic speed scaling* [29, 8, 16]. It has been observed that for most CMOS-based processors, the dynamic power consumption satisfies the cube-root rule, that is, the power of a processor is proportional to s^3 when it runs at speed s [8, 23]. Since the seminal paper by Yao, Demers and Shenker [31], most researchers, however, have assumed a more general power function s^α, where $\alpha > 1$ is the power parameter. As this power function is strictly convex, the total energy usage when executing a job can be significantly reduced by slowing down the speed of the processor at the expense of the job's performance. Thus, how to optimally tradeoff the conflicting objectives of energy and performance has become an active research topic in the algorithmic community. (See [19, 1] for two excellent surveys of the field.)

[*]School of Computer Engineering, Nanyang Technological University, Block N4, Nanyang Avenue, Singapore 639798. sunh0007@ntu.edu.sg

[†]Microsoft Research, Redmond, WA, USA 98052. yuxhe@microsoft.com

[‡]School of Computer Engineering, Nanyang Technological University, Block N4, Nanyang Avenue, Singapore 639798. hsu@ntu.edu.sg

In this paper, we focus on scheduling parallel jobs on multiprocessors with per-processor speed scaling capability [18, 21], that is, the speed of each processor can be individually scaled. A scheduling algorithm needs to have both a *processor allocation policy*, which decides the number of processors allocated to each job, and a *speed scaling policy*, which decides the speed of each allocated processor. Moreover, we assume that the parallel jobs under consideration can have time-varying parallelism. Thus, if the scheduling algorithm is not designed properly, it may incur large amount of energy waste when the jobs have relatively low parallelism, or cause severe execution delay and hence performance degradations when the parallelism of the jobs is high. This poses additional challenges to the speed scaling problem for parallel jobs compared with its sequential counterpart.

We adopt the objective function proposed by Albers and Fujiwara [2] that consists of a linear combination of overall energy consumption and some performance metric, which in this paper includes total flow time and makespan. The *flow time* of a job is defined to be the duration between its release and completion. The *total flow time* for a set of jobs is the sum of flow time of all jobs, and *makespan* is the completion time of the last completed job in the job set. Both total flow time and makespan are widely used performance metrics in scheduling literature: the former often measures the average response time of all users in the system while the latter is closely related to the throughput of the system. Although energy and flow time (or makespan) have different units, optimizing a linear combination of the two can be naturally interpreted by looking at both objectives from a unified point of view. Suppose that the user is willing to spend one unit of energy in order to reduce ρ units of total flow time (or makespan). Then, by changing the units of time and energy, we can assume without loss of generality that $\rho = 1$. Thus, the objective can be reduced to optimizing the total flow time (or makespan) plus energy for a set of jobs. In fact, minimizing sum of conflicting objectives is quite common in many bi-criteria optimization problems. In the scheduling literature, similar metrics have been considered previously that combine both performance and cost of scheduling as part of the objective functions [28, 26, 11].

Since Albers and Fujiwara [2] first proposed total flow time plus energy, many excellent results (see, e.g.,[4, 22, 3, 9, 10, 27, 30]) are obtained under different scheduling models. For instance, some results assume that the scheduling algorithm is *clairvoyant*, that is, it gains complete knowledge of a job, such as its total work, immediately upon the job's arrival; the other results are based on a more practical *non-clairvoyant* model, where the scheduler knows nothing about the un-executed portion of a job. Most of these results, however, are applicable to scheduling sequential jobs on a single processor, and to the best of our knowledge, no previous work is known that minimizes makespan plus energy. The closest results to ours are from Chan, Edmonds and Pruhs [10], and Sun, Cao and Hsu [27], who studied non-clairvoyant scheduling for parallel jobs on multiprocessors to minimize total flow time plus energy. In both work, it is observed that any non-clairvoyant algorithm that allocates one set of uniform-speed processors to a job performs poorly, or specifically $\Omega(P^{(\alpha-1)/\alpha^2})$-competitive, where P is the total number of processors. The intuition is that any non-clairvoyant algorithm may in the worst case allocate a "wrong" number of processors to a job compared to its parallelism, thus either incur excessive energy waste or cause severe execution delay.

Therefore, to obtain reasonable results, a non-clairvoyant algorithm need to be more flexible in assigning processors of different speeds to a job. To this end, Chan, Edmonds and Pruhs [10] assumed an execution model, in which each job can be executed by multiple groups of different speed processors. The execution rate of a job at any time is given by the fastest rate of all groups. They proposed a scheduling algorithm MultiLaps and showed that it is $O(\log P)$-competitive with respect to the total flow time plus energy. In addition, they also gave a lower bound of $\Omega(\log^{1/\alpha} P)$ for any non-clairvoyant algorithm. Sun, Cao and Hsu [27], on the other hand, assumed a different execution model, in which only one group of processors with different speeds are allocated to each job at any time, and the execution rate is determined by the speeds of the fastest processors that can be effectively utilized. They proposed algorithm N-Equi and showed that it is $O(\ln^{1/\alpha} P)$-competitive with respect to the total flow time plus energy on batched parallel jobs (i.e., all jobs are released at the same time). Both execution models are based on certain assumptions and can be justified in their respective terms. It is, however, quite difficult to predict which model is more practical to implement. In this paper, we first revisit non-clairvoyant scheduling for total flow time plus energy under the model by Sun, Cao and Hsu, and show that:

- N-Equi is $O(\ln P)$-competitive with respect to total flow time plus energy for parallel jobs with arbitrary release time, and any non-clairvoyant algorithm is $\Omega(\ln^{1/\alpha} P)$-competitive. Interestingly, both results match asymptotically those obtained under the execution model by Chan, Edmonds and Pruhs. The lower bound also suggests that N-Equi is asymptotically optimal in the batched setting.

Moreover in this paper, we consider a new scheduling model, which we call *semi-clairvoyant* model. Compared to the non-clairvoyant model, which does not allow a scheduler to have any knowledge about the un-executed portion of a job, we allow a semi-clairvoyant algorithm to know the available parallelism of a job at the immediate next step, or the *instantaneous parallelism*. Any future characteristic of the job, such as its remaining parallelism and work, is still unknown. In many parallel systems using centralized task queue or thread pool, instantaneous parallelism is simply the number of ready tasks in the queue or the number of ready threads in the pool, which is information practically available to the scheduler. Even for parallel systems using distributed scheduling such as work-stealing [6], instantaneous parallelism can also be collected or estimated through counting or sampling without introducing much system overhead. We first show that such semi-clairvoyance about the instantaneous parallelism of the jobs can bring significant performance improvement with respect to the total flow time plus energy. In particular,

- We present a semi-clairvoyant algorithm U-Ceq, and show that it is $O(1)$-competitive with respect to total flow time plus energy. This is the first $O(1)$-competitive result on multiprocessor speed scaling for parallel jobs.

Comparing to the performance of non-clairvoyant algorithms, the reason for the improvement is that upon knowing the instantaneous parallelism a semi-clairvoyant algorithm can now allocate a "right" number of processors to a job at any time, thus ensures that no energy will be wasted. At the same time, it can also guarantee sufficient execution rate by setting the total power consumption proportionally to the number of active jobs at any time and equally dividing it among the active jobs. This has been a common practice that intuitively provides the optimal balance between energy and total flow time [4, 22, 3, 9]. Moreover, unlike the best non-clairvoyant algorithm known so far [10, 27], which requires nonuniform speed scaling for an individual job, our semi-clairvoyant algorithm only requires allocating processors of uniform speed to a job, thus may have better feasibility in practice.

We also consider, for the first time in the literature, the objective of makespan plus energy. Unlike total flow time plus energy, where the completion time of each job contributes to the overall objective function, makespan is the completion time of the last job, and the other jobs only contribute to the energy consumption part of the objective, hence can be slowed down to improve the overall performance. However, without knowing the future information, such as the remaining work of the jobs, we show that it is harder to minimize makespan plus energy even in the semi-clairvoyant setting. Specifically,

- We present a semi-clairvoyant algorithm P-First and show that it is $O(\ln^{1-1/\alpha} P)$-competitive with respect to makespan plus energy for batched parallel jobs consisting of sequential and fully-parallel phases. We also give a matching lower bound of $\Omega(\ln^{1-1/\alpha} P)$ for any semi-clairvoyant algorithm.

In addition, compared to minimizing total flow time plus energy, where the common practice is to set the power proportionally to the number of active jobs, we show that the optimal strategy for minimizing makespan plus energy is to set the power consumption at a constant level, or more precisely $\frac{1}{\alpha-1}$ at any time, where α is the power parameter.

The rest this paper is organized as follows. Section 2 formally defines the models and the objective functions. Section 3 studies both non-clairvoyant and semi-clairvoyant scheduling for total flow time plus energy. Section 4 presents our semi-clairvoyant algorithm for makespan plus energy. Finally, Section 5 provides some discussions and future directions.

2 Models and Objective Functions

We model parallel jobs using time-varying parallelism profiles. Specifically, we consider a set $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ of n jobs to be scheduled on P processors. Adopting the notions in [14, 13, 15, 10], each job $J_i \in \mathcal{J}$ contains k_i phases $\langle J_i^1, J_i^2, \cdots, J_i^{k_i}\rangle$, and each phase J_i^k has an amount of *work* w_i^k and a *linear* speedup function Γ_i^k up to a certain parallelism h_i^k, where $h_i^k \geq 1$. Suppose that at any time t, job J_i is in its k-th phase and is allocated $a_i(t)$ processors, which may not have the same speed. We assume that the execution of the job at time t is then based on the *maximum utilization policy* [20, 5], which always utilizes faster processors before slower ones until the total number of utilized processors exceeds the parallelism of the job. In particular, let s_j denote the speed of the j-th allocated processor, and we can assume without loss of generality that $s_1 \geq s_2 \geq \cdots \geq s_{a_i(t)}$. Then, only $\bar{a}_i(t) = \min\{a_i(t), h_i^k\}$ fastest processors are effectively utilized, and the speedup or the execution rate of the job at time t is given by $\Gamma_i^k(a_i(t)) = \sum_{j=1}^{\bar{a}_i(t)} s_j$. The *span* l_i^k of phase J_i^k, which is a convenient parameter representing the time to execute the phase with h_i^k or more processors of unit speed, is then given by $l_i^k = w_i^k/h_i^k$. We say that phase J_i^k is *fully-parallel* if $h_i^k = \infty$ and it is *sequential* if $h_i^k = 1$. Moreover, if job J_i consists of only sequential and fully-parallel phases, we call it (Par-Seq)* job [25]. Finally, for each job J_i, we define its *total work* to be $w(J_i) = \sum_{k=1}^{k_i} w_i^k$ and define its *total span* to be $l(J_i) = \sum_{k=1}^{k_i} l_i^k$.

At any time t, a scheduling algorithm needs to specify the number $a_i(t)$ of processors allocated to each job J_i, as well as the speed of each allocated processor. We say that an algorithm is *non-clairvoyant* if it makes all scheduling decisions without any current and future information of the jobs, such as their release time, parallelism profile and remaining work. In addition, we say that an algorithm is *semi-clairvoyant* if it is only aware of the current parallelism or *instantaneous parallelism* of the jobs, but not their future parallelism and remaining work. We require that the total processor allocations cannot be more than the total number of processors at any time in a valid schedule, i.e., $\sum_{i=1}^{n} a_i(t) \leq P$. Let

r_i denote the *release time* of job J_i. If all jobs are released together in a single *batch*, then their release time can be assumed to be all 0. Otherwise, we can assume without loss of generality that the first released job arrives at time 0. Let c_i denote the completion time of job J_i. We also require that a valid schedule must complete all jobs in finite amount of time and cannot begin to execute a phase of a job unless it has completed all its preceding phases, i.e., $r_i = c_i^0 \leq c_i^1 \leq \cdots \leq c_i^{k_i} = c_i < \infty$, and $\int_{c_i^{k-1}}^{c_i^k} \Gamma_i^k(a_i(t))dt = w_i^k$ for all $1 \leq k \leq k_i$, where c_i^k denotes the completion time of phase J_i^k.

The *flow time* f_i of any job J_i is the duration between its completion and release, i.e., $f_i = c_i - r_i$. The *total flow time* $F(\mathcal{J})$ of all jobs in $\mathcal{J}$ is given by $F(\mathcal{J}) = \sum_{i=1}^n f_i$, and the makespan $M(\mathcal{J})$ is the completion time of the last completed job, i.e., $M(\mathcal{J}) = \max_{i=1,\cdots,n} c_i$. Job J_i is said to be *active* at time t if it is released but not completed at t, i.e., $r_i \leq t \leq c_i$. An alternative expression for the total flow time is $F(\mathcal{J}) = \int_0^\infty n_t dt$, where n_t is the number of active jobs at time t. For each processor at a particular time, its power is given by s^α if it runs at speed s, where $\alpha > 1$ is the power parameter. Hence, if a processor is not allocated to any job, we can set its speed to 0, so it does not consume any power. Let $u_i(t)$ denote the power consumed by job J_i at time t, i.e., $u_i(t) = \sum_{j=1}^{a_i(t)} s_j^\alpha$. The overall energy consumption e_i of the job is given by $e_i = \int_0^\infty u_i(t)dt$, and the total energy consumption $E(\mathcal{J})$ of the job set is $E(\mathcal{J}) = \sum_{i=1}^n e_i$, or alternatively $E(\mathcal{J}) = \int_0^\infty u_t dt$, where $u_t = \sum_{i=1}^n u_i(t)$ denotes the total power consumption of all jobs at time t. In this paper, we consider total flow time plus energy $G(\mathcal{J})$ and makespan plus energy $H(\mathcal{J})$ of the job set, i.e., $G(\mathcal{J}) = F(\mathcal{J}) + E(\mathcal{J})$ and $H(\mathcal{J}) = M(\mathcal{J}) + E(\mathcal{J})$. The objective is to minimize either $G(\mathcal{J})$ or $H(\mathcal{J})$.

We use *competitive analysis* [7] to evaluate an online scheduling algorithm by comparing its performance with that of an optimal offline scheduler. An online algorithm A is said to be c_1-*competitive* with respect to total flow time plus energy if $G_A(\mathcal{J}) \leq c_1 \cdot G^*(\mathcal{J})$ for any job set $\mathcal{J}$, where $G^*(\mathcal{J})$ denotes the total flow time plus energy of $\mathcal{J}$ under an optimal offline scheduler. Similarly, an online algorithm B is said to be c_2-*competitive* with respect to makespan plus energy if for any job set $\mathcal{J}$ we have $H_B(\mathcal{J}) \leq c_2 \cdot H^*(\mathcal{J})$, where $H^*(\mathcal{J})$ denotes the makespan plus energy of the job set under an optimal offline scheduler.

3 Total Flow Time Plus Energy

We consider the objective of total flow time plus energy in this section. We first revisit the non-clairvoyant algorithm N-Equi [27] by showing its competitive ratio for arbitrary released jobs. We then derive a lower bound on the competitive ratio of any non-clairvoyant algorithm. Finally, we present a semi-clairvoyant algorithm U-Ceq and show that it significantly improves upon any non-clairvoyant algorithm.

3.1 Preliminaries

We first derive a lower bound on the total flow time plus energy of any scheduler, which will help us conveniently bound the performance of the online algorithms through indirect comparison instead of comparing directly with the optimal.

Lemma 1 *The total flow time plus energy of any set $\mathcal{J}$ of n jobs under the optimal scheduler satisfies* $G^*(\mathcal{J}) \geq G_1^*(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^n \sum_{k=1}^{k_i} \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}$.

Proof. Consider any phase J_i^k of job J_i. The optimal scheduler will only perform better if there is an unlimited number of processors at its disposal. In this case, it will allocate a processors of the same speed, say s, to the phase throughout its execution, since by the convexity of the power function, if different speeds are used, then averaging the speeds will result in the same execution rate but less energy consumed [31]. Moreover, we have $a \le h_i^k$, since allocating more processors to a phase than its parallelism will incur more energy without improving flow time. The flow time plus energy introduced by the execution of J_i^k is then given by $\frac{w_i^k}{as} + \frac{w_i^k}{as} \cdot as^\alpha = w_i^k \left(\frac{1}{as} + s^{\alpha-1}\right) \ge \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w_i^k}{a^{1-1/\alpha}} \ge \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}$. Extending this property over all phases and all jobs gives the lower bound. □

We now outline the *amortized local competitiveness argument* [4] to prove the competitive ratio of any online scheduling algorithm A. We first define some notations. For any job set $\mathcal{J}$ at time t, let $\frac{dG_{\mathrm{A}}(\mathcal{J}(t))}{dt}$ denote the rate of change for flow time plus energy under online algorithm A, and let $\frac{dG^*(\mathcal{J}^*(t))}{dt}$ denote the rate of change for flow time plus energy under the optimal. Apparently, we have $\frac{dG_{\mathrm{A}}(\mathcal{J}(t))}{dt} = n_t + u_t$, and $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + u_t^*$, where n_t^* and u_t^* denote the number of active jobs and the power under the optimal at time t. Moreover, we let $\frac{dG_1^*(\mathcal{J}(t))}{dt}$ denote the rate of change for the lower bound given in Lemma 1 with respect to the execution of the job set under A at time t. We also need to define a potential function $\Phi(t)$ associated with the status of the job set at any time t under both the online algorithm and the optimal. Then, we can similarly define $\frac{d\Phi(t)}{dt}$ to be the rate of change for the potential function at t. The following lemma shows that the competitive ratio of algorithm A can be obtained by bounding the instantaneous performance of A at any time t with respect to the optimal scheduler through these rates of change.

Lemma 2 *Suppose that an online algorithm* A *schedules a set* $\mathcal{J}$ *of jobs. Then* A *is* $(c_1 + c_2)$*-competitive with respect to total flow time plus energy, if given a potential function* $\Phi(t)$*, the execution of the job set under* A *satisfies*

- *Boundary condition:* $\Phi(0) \le 0$ *and* $\Phi(\infty) \ge 0$*;*
- *Arrival condition:* $\Phi(t)$ *does not increase when a new job arrives;*
- *Completion condition:* $\Phi(t)$ *does not increase when a job completes under either* A *or the optimal offline scheduler;*
- *Running condition:* $\frac{dG_{\mathrm{A}}(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \le c_1 \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + c_2 \cdot \frac{dG_1^*(\mathcal{J}(t))}{dt}$.

Proof. Let T denote the set of time instances when a job arrives or completes under either the online algorithm A or the optimal offline scheduler. Integrating the running condition over time, we get $G_A(\mathcal{J}) + \Phi(\infty) - \Phi(0) + \sum_{t\in T}(\Phi(t^-) - \Phi(t^+)) \le c_1 \cdot G^*(\mathcal{J}) + c_2 \cdot G_1^*(\mathcal{J})$, where t^- and t^+ denote the time instances right before and after time t. Now, applying boundary, arrival and completion conditions to the above inequality, we get $G_A(\mathcal{J}) \le c_1 \cdot G^*(\mathcal{J}) + c_2 \cdot G_1^*(\mathcal{J})$. Since $G_1^*(\mathcal{J})$ is a lower bound on the total flow time plus energy of job set $\mathcal{J}$ according to Lemma 1, the performance of algorithm A thus satisfies $G_A(\mathcal{J}) \le (c_1 + c_2) \cdot G^*(\mathcal{J})$. □

3.2 Non-clairvoyant Algorithm: N-EQUI

In this subsection, we revisit the non-clairvoyant algorithm N-Equi (Nonuniform Equipartition) [27], which is described in Algorithm 1. The idea of N-Equi is that at any time it allocates an equal share P/n_t of processors to each active job, and the speeds of the allocated processors are set monotonically decreasing according to a scaled version of harmonic series. We assume that the processor allocation P/n_t is always an integer, otherwise by rounding it to $\lfloor P/n_t \rfloor$, the bounds derived will increase by at most a constant factor.

Algorithm 1 N-EQUI (at any time t)

```
1: allocate $a_i(t) = P/n_t$ processors to each active job $J_i$,
2: set the speed of the $j$-th allocated processor to job $J_i$ as $s_{ij}(t) = \left(\frac{1}{(\alpha-1)H_P \cdot j}\right)^{1/\alpha}$, where
   $1 \leq j \leq a_i(t)$ and $H_P = 1 + \frac{1}{2} + \cdots + \frac{1}{P}$ is $P$-th harmonic number.
```

At time t, when job J_i is in its k-th phase, we say that it is *satisfied* if its processor allocation is at least the instantaneous parallelism, i.e., $a_i(t) \geq h_i^k$. Otherwise, the job is *deprived* if $a_i(t) < h_i^k$. Let $\mathcal{J}_S(t)$ and $\mathcal{J}_D(t)$ denote the sets of satisfied and deprived jobs at time t, respectively. For convenience, we let $n_t^S = |\mathcal{J}_S(t)|$ and $n_t^D = |\mathcal{J}_D(t)|$. Since a job is either satisfied or deprived, we have $n_t = n_t^S + n_t^D$. Moreover, we define $x_t = n_t^D/n_t$ to be the *deprived ratio.* Let $\bar{a}_i(t) = \min\{a_i(t), h_i^k\}$. By approximating summations with integrals, the execution rate of job J_i can be shown to satisfy $\left(\frac{1}{(\alpha-1)H_P}\right)^{1/\alpha} \frac{\bar{a}_i(t)^{1-1/\alpha}}{2^{1/\alpha}} \leq \Gamma_i^k(a_i(t)) \leq \left(\frac{1}{(\alpha-1)H_P}\right)^{1/\alpha} \frac{\bar{a}_i(t)^{1-1/\alpha}}{1-1/\alpha}$ at time t. Moreover, the power consumption of job J_i satisfies $u_i(t) \leq \frac{1}{\alpha-1}$, and hence the overall power consumption satisfies $u_t \leq \frac{n_t}{\alpha-1}$.

To bound the performance of N-EQUI, we adopt the potential function by Lam et al. [22] in the analysis of online speed scaling algorithm for sequential jobs. Specifically, we define $n_t(z)$ to be the number of active jobs whose remaining work is at least z at time t under N-EQUI, and define $n_t^*(z)$ to be the number of active jobs whose remaining work is at least z under the optimal. The potential function is defined to be

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) - n_t(z)^{1-1/\alpha} n_t^*(z) \right] dz, \tag{1}$$

where $\eta = \eta' \frac{H_P^{1/\alpha}}{P^{1-1/\alpha}}$ and η' is a constant to be specified later. With the help of Lemma 2, the competitive ratio of N-EQUI is proved in the following theorem.

Theorem 3 N-EQUI *is $O(\ln P)$-competitive with respect to the total flow time plus energy for any set of parallel jobs, where P is the total number of processors.*

Proof. We will show that the execution of any job set under N-EQUI (NE for short) satisfies the boundary, arrival and completion conditions, as well as the running condition $\frac{dG_{\mathrm{NE}}(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \leq c_1 \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + c_2 \cdot \frac{dG_1^*(\mathcal{J}(t))}{dt}$, where $c_1 = O(\ln P)$ and $c_2 = O(\ln^{1/\alpha} P)$. Then the theorem is directly implied.

- *Boundary condition*: at time 0, no jobs exist, so $n_t(z)$ and $n_t^*(z)$ are 0 for all z. Hence, $\Phi(0) = 0$. At time ∞, all jobs are completed, so again $\Phi(\infty) = 0$.

- *Arrival condition*: Let t^- and t^+ denote the instances right before and after a new job with work w arrives at time t. Hence, we have $n_{t^+}(z) = n_{t^-}(z) + 1$ for $z \leq w$ and $n_{t^+}(z) = n_{t^-}(z)$ for $z > w$, and similarly $n_{t^+}^*(z) = n_{t^-}^*(z) + 1$ for $z \leq w$ and $n_{t^+}^*(z) = n_{t^-}^*(z)$ for $z > w$. For convenience, we define $\phi_t(z) = \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha}\right) - n_t(z)^{1-1/\alpha} n_t^*(z)$. It is obvious that for $z > w$, we have $\phi_{t^+}(z) = \phi_{t^-}(z)$. For $z \leq w$, we can get $\phi_{t^+}(z) -$

$\phi_{t^-}(z) = n^*_{t^-}(z)\left(n_{t^-}(z)^{1-1/\alpha} - (n_{t^-}(z)+1)^{1-1/\alpha}\right) \le 0$. Hence, $\Phi(t^+) = \eta\int_0^\infty \phi_{t^+}(z)dz \le \eta\int_0^\infty \phi_{t^-}(z)dz = \Phi(t^-)$.

- *Completion condition*: when a job completes under either N-EQUI or the optimal, $\Phi(t)$ is unchanged since $n(t)$ or $n^*(t)$ is unchanged for all $z > 0$.

- *Running condition*: At any time t, suppose that the optimal offline scheduler sets the speed of the j-th processor to s^*_j. We have $\frac{dG_{\mathrm{NE}}(\mathcal{J}(t))}{dt} = n_t + u_t \le \frac{\alpha}{\alpha-1}n_t$ and $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n^*_t + u^*_t = n^*_t + \sum_{j=1}^P \left(s^*_j\right)^\alpha$. To bound the rate of change $\frac{dG^*_1(\mathcal{J}(t))}{dt}$, we consider each satisfied job $J_i \in J_S(t)$. Suppose that at time t, J_i is in its k-th phase under N-EQUI, then the execution rate of the job is given by $\Gamma^k_i(a_i(t)) \ge \left(\frac{1}{(\alpha-1)H_P}\right)^{1/\alpha}\frac{\left(h^k_i\right)^{1-1/\alpha}}{2^{1/\alpha}}$. Since $\frac{dG^*_1(\mathcal{J}(t))}{dt}$ only depends on the parts of the jobs that are executed by N-EQUI at time t, we have $\frac{dG^*_1(\mathcal{J}(t))}{dt} \ge \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\sum_{J_i\in\mathcal{J}_S(t)}\frac{\Gamma^k_i(a_i(t))}{\left(h^k_i\right)^{1-1/\alpha}} \ge \frac{\alpha}{\alpha-1}\left(\frac{1}{2H_P}\right)^{1/\alpha} n^S_t = \frac{\alpha}{\alpha-1}\left(\frac{1}{2H_P}\right)^{1/\alpha}(1-x_t)n_t$.

Now, we focus on finding an upper bound on the rate of change $\frac{d\Phi(t)}{dt}$ for the potential function $\Phi(t)$ at time t. In particular, we consider the set $\mathcal{J}_D(t)$ of deprived jobs. In the worst case, the n^D_t deprived jobs may have the most remaining work. Again, we assume that at time t job $J_i \in \mathcal{J}_D(t)$ is in its k-th phase under N-EQUI. The change of the potential function can then be bounded by

$$\begin{aligned}
\frac{d\Phi(t)}{dt} &\le \frac{\eta}{dt}\int_0^\infty \left[\left(\sum_{i=1}^{n_{t+dt}(z)} i^{1-1/\alpha}\right) - \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha}\right)\right] dz \\
&\quad + \frac{\eta}{dt}\int_0^\infty \left[n_t(z)^{1-1/\alpha}\left(n^*_t(z) - n^*_{t+dt}(z)\right) + n^*_t(z)\left(n_t(z)^{1-1/\alpha} - n_{t+dt}(z)^{1-1/\alpha}\right)\right] dz \\
&\le \frac{\eta' H_P^{1/\alpha}}{P^{1-1/\alpha}}\left(-\sum_{i=1}^{n^D_t} i^{1-1/\alpha}\cdot\Gamma^k_i(a_i(t)) + n_t^{1-1/\alpha}\sum_{j=1}^P s^*_j + n^*_t\sum_{i=1}^{n_t}\left(i^{1-1/\alpha} - (i-1)^{1-1/\alpha}\right)\Gamma^k_i(a_i(t))\right).
\end{aligned}$$

We can get $\sum_{i=1}^{n^D_t} i^{1-1/\alpha} \ge \int_0^{n^D_t} i^{1-1/\alpha}di = \frac{\left(n^D_t\right)^{2-1/\alpha}}{2-1/\alpha} \ge \frac{x_t^2 n_t^{2-1/\alpha}}{2}$ and $\sum_{i=1}^{n_t}\left(i^{1-1/\alpha} - (i-1)^{1-1/\alpha}\right) = n_t^{1-1/\alpha}$. Moreover, according to Lemma 4, we have $n_t^{1-1/\alpha}\sum_{j=1}^P s^*_j \le \frac{\lambda(H_P\cdot P)^{1-1/\alpha}}{\alpha}\sum_{j=1}^P\left(s^*_j\right)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}(H_P\cdot P)^{1/\alpha}}Pn_t$, where λ is a constant to be specified later. Substituting these bounds as well as the upper and lower bounds of $\Gamma^k_i(a_i(t))$ into $\frac{d\Phi(t)}{dt}$ and simplify, we have

$$\frac{d\Phi(t)}{dt} \le \eta'\left(-\frac{x_t^2}{4(\alpha-1)^{1/\alpha}}n_t + \frac{\lambda H_P}{\alpha}\sum_{j=1}^P\left(s^*_j\right)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}}n_t + \frac{\alpha}{(\alpha-1)^{1+1/\alpha}}n^*_t\right). \quad (2)$$

Now, we set $\eta' = \frac{4\alpha^2}{(\alpha-1)^{1-1/\alpha}}$ and $\lambda = 4^{\alpha-1}(\alpha-1)^{1-1/\alpha}$. Substituting Inequality (3) as well as the rates of change $\frac{dG_{\mathrm{NE}}(\mathcal{J}(t))}{dt}$, $\frac{dG^*(\mathcal{J}^*(t))}{dt}$ and $\frac{dG^*_1(\mathcal{J}(t))}{dt}$ into the running condition, we can see that in order to satisfy it for all values of x_t, the multipliers c_1 and c_2 can be set to $c_1 = \max\{\frac{4\alpha^3}{(\alpha-1)^2}, 4^\alpha\alpha H_P\}$ and $c_2 = 2\alpha\cdot(2H_P)^{1/\alpha}$. Since α can be considered as a constant, and it is well-known that $H_P = O(\ln P)$, the theorem is proved. □

Lemma 4 *For any $n_t \ge 0$, $s^*_j \ge 0$ and $\lambda > 0$, we have that $n_t^{1-1/\alpha}s^*_j \le \frac{\lambda(H_P\cdot P)^{1-1/\alpha}}{\alpha}\left(s^*_j\right)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}(H_P\cdot P)^{1/\alpha}}n_t$.*

Proof. The lemma is a direct result of Young's Inequality [17], which is stated formally as follows. If f is a continuous and strictly increasing function on $[0, c]$ with $c > 0$, $f(0) = 0$, $a \in [0, c]$ and $b \in [0, f(c)]$, then $ab \leq \int_0^a f(x)dx + \int_0^b f^{-1}(x)dx$, where f^{-1} is the inverse function of f. By setting $f(x) = \lambda (H_P \cdot P)^{1-1/\alpha} x^{\alpha-1}$, $a = s_j^*$ and $b = n_t^{1-1/\alpha}$, the lemma is directly implied. $\square$

3.3 Lower Bound of Non-clairvoyant Algorithm

In this subsection, we prove a lower bound on the competitive ratio of any deterministic non-clairvoyant algorithm. In particular, this lower bound matches asymptotically the upper bound of N-EQUI for batched parallel jobs [27], hence suggests that N-EQUI is asymptotically optimal in the batched setting.

Theorem 5 *Any deterministic non-clairvoyant algorithm is $\Omega(\ln^{1/\alpha} P)$-competitive with respect to the total flow time plus energy, where P is the total number of processors.*

Proof. Consider a job set $\mathcal{J}$ of a single job with constant parallelism h and work w, where $1 \leq h \leq P$ and $w > 0$. For any non-clairvoyant algorithm A, we can assume without loss of generality that it allocates all P processors to the job with speeds satisfying $s_1 \geq s_2 \geq \cdots \geq s_P \geq 0$, which do not change throughout the execution. Let $u = \sum_{j=1}^P s_j^\alpha$ denote the power of A at any time. The flow time plus energy of $\mathcal{J}$ scheduled by A is $G_A(\mathcal{J}) = (1 + u) \frac{w}{\sum_{j=1}^h s_j}$. The optimal offline scheduler, knowing the parallelism h, will allocate exactly h processors of speed $\left(\frac{1}{(\alpha-1)h}\right)^{1/\alpha}$, thus incurring flow time plus energy of $G^*(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w}{h^{1-1/\alpha}}$. The competitive ratio of A is $\frac{G_{\mathrm{A}}(\mathcal{J})}{G^*(\mathcal{J})} = \frac{(\alpha-1)^{1-1/\alpha}(1+u)}{\alpha} \cdot \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$.

The adversary will choose parallelism h to maximize this ratio, i.e., to find $\max_{1 \leq h \leq P} \frac{G_{\mathrm{A}}(\mathcal{J})}{G^*(\mathcal{J})}$, while the online algorithm A chooses $(s_1, \cdots, s_P)$ to minimize $\max_{1 \leq h \leq P} \frac{G_{\mathrm{A}}(\mathcal{J})}{G^*(\mathcal{J})}$ regardless of the choice of h. According to Lemma 6, $\max_{1 \leq h \leq P} \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$ is minimized when $\frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j} = \frac{(h-1)^{1-1/\alpha}}{\sum_{j=1}^{h-1} s_j}$ for $h = 2, \cdots, P$. Hence, the best non-clairvoyant algorithm will set $s_j = \left(j^{1-1/\alpha} - (j-1)^{1-1/\alpha}\right) s_1$ for $j = 1, 2, \cdots, P$. Since $j^{1-1/\alpha} - (j-1)^{1-1/\alpha} \geq \frac{1-1/\alpha}{j^{1/\alpha}}$, we have $s_j \geq \frac{1-1/\alpha}{j^{1/\alpha}} s_1$. Substituting them into $u = \sum_{j=1}^P (s_j)^\alpha$, we get $s_1 \leq \frac{\alpha u^{1/\alpha}}{(\alpha-1) H_P^{1/\alpha}}$. The competitive ratio of any non-clairvoyant algorithm satisfies $\frac{G_{\mathrm{A}}(\mathcal{J})}{G^*(\mathcal{J})} \geq \frac{(\alpha-1)^{1-1/\alpha}(1+u)}{\alpha s_i} \geq \frac{(\alpha-1)^{2-1/\alpha}}{\alpha^2} \cdot \frac{1+u}{u^{1/\alpha}} H_P^{1/\alpha} \geq \frac{\alpha-1}{\alpha} H_P^{1/\alpha}$. The last inequality is because $\frac{1+u}{u^{1/\alpha}}$ is minimized when $u = \frac{1}{\alpha-1}$. Since $H_P = \Omega(\ln P)$, the theorem is proved. $\square$

Lemma 6 *For any $P \geq 1$, $\alpha > 1$ and $b > 0$, subject to the condition that $\sum_j^P s_j^\alpha = b$ and $s_1 \geq s_2 \geq \cdots \geq s_P \geq 0$, $\max_{1 \leq h \leq P} \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$ is minimized when $(s_1, s_2, \cdots, s_P)$ satisfy $\frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j} = \frac{(h-1)^{1-1/\alpha}}{\sum_{j=1}^{h-1} s_j}$ for all $h = 2, \cdots, P$.*

Proof. The proof is in Appendix A. $\square$

3.4 Semi-clairvoyant Algorithm: U-CEQ

We now present our semi-clairvoyant scheduling algorithm U-CEQ (Uniform Conservative EQUI) and analyze its total flow time plus energy. In particular, we show that semi-clairvoyance makes a big difference on the performance of an online algorithm by proving that U-CEQ achieves $O(1)$-competitive. As shown in Algorithm 2, U-CEQ at any time t works similarly to N-EQUI in terms of processor allocation, except that it never allocates more processors than a job's instantaneous parallelism h_i^k. Moreover, the speed of all processors allocated to a job in U-CEQ is set in a uniform manner.

Algorithm 2 U-CEQ (at any time t)

```
1: allocate $a_i(t) = \min\{h_i^k, P/n_t\}$ processors to each active job $J_i$,
2: set the speed of all allocated processors to job $J_i$ as $s_i(t) = \left(\frac{1}{(\alpha-1)a_i(t)}\right)^{1/\alpha}$.
```

Again, we say that active job J_i is *satisfied* at time t if $a_i(t) = h_i^k$, and that it is *deprived* if $a_i(t) < h_i^k$. We can see that job J_i at time t scheduled by U-CEQ has execution rate $\Gamma_i^k(a_i(t)) = \frac{a_i(t)^{1-1/\alpha}}{(\alpha-1)^{1/\alpha}}$ and consumes power $u_i(t) = \frac{1}{\alpha-1}$. Therefore, the overall power consumption is $u_t = \frac{n_t}{\alpha-1}$. Since there is no energy waste, we will show that this execution rates is sufficient to ensure the competitive performance of the U-CEQ algorithm.

Theorem 7 U-CEQ *is $O(1)$-competitive with respect to the total flow time plus energy for any set of parallel jobs.*

Proof. As with N-EQUI, we prove the $O(1)$-competitiveness of U-CEQ using amortized local competitiveness argument with the same potential function given in Eq. (1), but η is now set to $\eta = \frac{\eta'}{P^{1-1/\alpha}}$ and $\eta' = \frac{2\alpha^2}{(\alpha-1)^{1-1/\alpha}}$. Apparently, the boundary, arrival and completion conditions hold regardless of the scheduling algorithm. We need only show that the execution of any job set under U-CEQ (UC for short) satisfies the running condition $\frac{dG_{\mathrm{UC}}(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \le c_1 \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + c_2 \cdot \frac{dG_1^*(\mathcal{J}(t))}{dt}$, where c_1 and c_2 are both constants with respect to P.

Following the proof of Theorem 3, we have $\frac{dG_{\mathrm{UC}}(\mathcal{J}(t))}{dt} = \frac{\alpha}{\alpha-1}n_t$, $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + \sum_{j=1}^{P}\left(s_j^*\right)^\alpha$, and $\frac{dG_1^*(\mathcal{J}(t))}{dt} \ge \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\sum_{J_i\in\mathcal{J}_S(t)}\frac{\Gamma_i^k(a_i(t))}{\left(h_i^k\right)^{1-1/\alpha}} = \frac{\alpha}{\alpha-1}(1-x_t)n_t$. Moreover, the rate of change $\frac{d\Phi(t)}{dt}$ for the potential function $\Phi(t)$ at time t can be shown to satisfy

$$\begin{aligned}\frac{d\Phi(t)}{dt} &\le \frac{\eta'}{P^{1-1/\alpha}}\left(-\sum_{i=1}^{n_t^D} i^{1-1/\alpha}\cdot\Gamma_i^k(a_i(t)) + n_t^{1-1/\alpha}\sum_{j=1}^{P} s_j^* + n_t^*\sum_{i=1}^{n_t}\left(i^{1-1/\alpha} - (i-1)^{1-1/\alpha}\right)\Gamma_i^k(a_i(t))\right)\\ &\le \eta'\left(-\frac{x_t^2}{2(\alpha-1)^{1/\alpha}}n_t + \frac{\lambda}{\alpha}\sum_{j=1}^{P}\left(s_j^*\right)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}}n_t + \frac{n_t^*}{(\alpha-1)^{1/\alpha}}\right),\end{aligned}$$

where $\lambda = 2^{\alpha-1}(\alpha-1)^{1-1/\alpha}$. Substituting these bounds into the running condition, we can see that in order to satisfy it for all values of x_t, we can set $c_1 = \max\{\frac{2\alpha^2}{\alpha-1}, 2^\alpha\alpha\}$ and $c_2 = 2\alpha$, which are both constants in terms of P. Hence, the theorem is proved. □

We can see that U-CEQ significantly improves upon any non-clairvoyant algorithm with respect to the total flow time plus energy, which is essentially a result of not wasting any energy yet still guaranteeing sufficient execution rates for the jobs. Since we know that non-clairvoyant algorithms perform similarly to semi-clairvoyant ones with respect to the total flow time alone [12, 14], it reveals the importance of (even partial) clairvoyance when energy is also of concern.

Moreover, since U-Ceq takes advantage of the parallelism information of a job, uniform speed scaling is sufficient to ensure its competitiveness. Therefore, compared to the non-clairvoyant algorithm N-Equi that requires nonuniform speed scaling, U-Ceq may find better feasibility in practice. It is not hard to see, however, that non-uniform speed scaling is not beneficial in the semi-clairvoyant setting. Instead, it will degrade the performance, since generally less energy will be consumed at the same execution rate with uniform speed.

4 Makespan Plus Energy

In this section, we consider the objective of makespan plus energy. In particular, we propose a semi-clairvoyant algorithm P-First (Parallel-First) and show that it is $O(\ln^{1-1/\alpha} P)$-competitive for any set of batched (Par-Seq)* jobs. We also show that this ratio is asymptotically optimal for any semi-clairvoyant algorithm.

4.1 Performances of the Optimal

We first show that as far as minimizing makespan plus energy for batched jobs, the optimal (online/offline) strategy maintains a constant total power of $\frac{1}{\alpha-1}$ at any time. This corresponds to the *power equality property* shown in [24], which applies to any optimal offline algorithm for the makespan minimization problem with an energy budget.

Lemma 8 *For any schedule* A *on a set* $\mathcal{J}$ *of batched jobs, there exists a schedule* B *that executes the same set of jobs with a constant total power of* $\frac{1}{\alpha-1}$ *at any time, and performs no worse than* A *with respect to makespan plus energy, i.e.,* $H_{\rm B}(\mathcal{J}) \leq H_{\rm A}(\mathcal{J})$.

Proof. For any schedule A on a set $\mathcal{J}$ of batched jobs, consider an interval Δt during which the speeds of all processors, denoted as $(s_1, s_2, \cdots, s_P)$, remain unchanged. The makespan plus energy of A incurred by executing this portion of the job set is given by $H_{\rm A} = \Delta t(1+u)$, where $u = \sum_{j=1}^{P} s_j^\alpha$ is the power consumption of all processors during Δt. We now construct schedule B in such a way that it executes the same portion of the job set by running the j-th processor at speed $k \cdot s_j$, where $k = \left(\frac{1}{(\alpha-1)u}\right)^{1/\alpha}$. This portion will then finish under schedule B in $\frac{\Delta t}{k}$ time, and the power consumption at any time during this interval is given by $\frac{1}{\alpha-1}$. The makespan plus energy of B incurred by executing the same portion of the job set is $H_{\rm B} = \frac{\Delta t}{k}(1 + \frac{1}{\alpha-1}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\Delta t u^{1/\alpha}$. Since $\frac{1+u}{u^{1/\alpha}}$ is minimized when $u = \frac{1}{\alpha-1}$, we have $\frac{H_A}{H_B} = \frac{(\alpha-1)^{1-1/\alpha}}{\alpha} \cdot \frac{1+u}{u^{1/\alpha}} \geq 1$, i.e., $H_A \geq H_B$. Extending the same argument to all such intervals in schedule A proves the lemma. $\square$

Compared to total flow time plus energy, where the completion time of each job contributes to the overall objective function, makespan for a set of jobs is the completion time of the last job. In this case, the other jobs only contribute to the energy consumption part of the objective, thus can be slowed down to consume less energy and eventually lead to better overall performance. Based on this observation as well as the result of Lemma 8, we derive the performance of the optimal offline scheduler for any batched (Par-Seq)* job set in the following lemma.

Lemma 9 *The optimal makespan plus energy of any batched set $\mathcal{J}$ of* (PAR-SEQ)* *jobs satisfies* $H^*(\mathcal{J}) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \max\{\frac{\sum_{i=1}^{n} w(J_i)}{P^{1-1/\alpha}}, (\sum_{i=1}^{n} l(J_i)^{\alpha})^{1/\alpha}\}$.

Proof. Given any job $J_i \in \mathcal{J}$, define $J_{i,P}$ to be a job with a single fully-parallel phase of the same work as J_i, and define $J_{i,S}$ to be a job with a single sequential phase of the same span as J_i. Moreover, we define job set $\mathcal{J}_P$ to be $\mathcal{J}_P = \{J_{i,P} : J_i \in \mathcal{J}\}$ and define $\mathcal{J}_S$ to be $\mathcal{J}_S = \{J_{i,S} : J_i \in \mathcal{J}\}$. Clearly, the optimal makespan plus energy for $\mathcal{J}_P$ and $\mathcal{J}_S$ will be no worse than that for the original job set $\mathcal{J}$, i.e., $H^*(\mathcal{J}) \geq H^*(\mathcal{J}_P)$ and $H^*(\mathcal{J}) \geq H^*(\mathcal{J}_S)$, since the optimal schedule for $\mathcal{J}$ is a valid schedule for $\mathcal{J}_P$ and $\mathcal{J}_S$.

For job set $\mathcal{J}_P$, the optimal scheduler can execute the jobs in any order since all jobs are fully-parallel in this case. Moreover, by the convexity of the power function, all P processors are run with constant speed s. According to Lemma 8, we have $Ps^{\alpha} = \frac{1}{\alpha-1}$, hence $s = \left(\frac{1}{(\alpha-1)P}\right)^{1/\alpha}$. The makespan plus energy is therefore $H^*(\mathcal{J}_P) = \frac{\sum_{i=1}^{n} w(J_i)}{Ps}(1 + Ps^{\alpha}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{\sum_{i=1}^{n} w(J_i)}{P^{1-1/\alpha}}$.

For job set $\mathcal{J}_S$, allowing the optimal scheduler to have at least $\max\{n, P\}$ processors can only improve its performance. In this case, the optimal will execute each job on a single processor with constant speed. Moreover, all jobs are completed simultaneously, since otherwise jobs completed earlier can be slowed down to save energy without affecting makespan. Let s_i denote the speed by the optimal for job $J_{i,S}$, so $\frac{l(J_1)}{s_1} = \frac{l(J_2)}{s_2} = \cdots = \frac{l(J_n)}{s_n}$, and $\sum_{i=1}^{n} s_i^{\alpha} = \frac{1}{\alpha-1}$ according to Lemma 8. Therefore, the speeds satisfy $s_i = \frac{1}{(\alpha-1)^{1/\alpha}} \cdot \frac{l(J_i)}{\left(\sum_{i=1}^{n} l(J_i)^{\alpha}\right)^{1/\alpha}}$ for $i = 1, 2, \cdots, n$. The makespan plus energy is $H^*(\mathcal{J}_S) = \frac{l(J_1)}{s_1} + \frac{l(J_1)}{s_1}\left(\sum_{i=1}^{n} s_i^{\alpha}\right) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\left(\sum_{i=1}^{n} l(J_i)^{\alpha}\right)^{1/\alpha}$. □

4.2 Semi-clairvoyant Algorithm: P-FIRST

We now present a semi-clairvoyant algorithm P-FIRST (Parallel-First) for any batched set $\mathcal{J}$ of (PAR-SEQ)* jobs. Basically, P-FIRST will first execute the fully-parallel phases of any job whenever possible, and then executes the sequential phases of all jobs at the same rate. Specifically, at any time t when there are n_t active jobs, P-FIRST works as shown in Algorithm 3.

Algorithm 3 P-FIRST

```
1: if there is at least one active job in fully-parallel phase at any time $t$ then
2:     execute any such job on $P$ processors; each processor runs at speed $\left(\frac{1}{(\alpha-1)P}\right)^{1/\alpha}$.
3: else
4:     execute all active jobs on $P' = \min\{n_t, P\}$ processors by equally sharing the processors among the jobs; each processor runs at speed $\left(\frac{1}{(\alpha-1)P'}\right)^{1/\alpha}$.
5: end if
```

As we can see, P-FIRST ensures that the overall energy consumption $E(\mathcal{J})$ and the makespan $M(\mathcal{J})$ of job set $\mathcal{J}$ satisfies $E(\mathcal{J}) = \frac{1}{\alpha-1}M(\mathcal{J})$, since at any time t, the total power is given by $u_t = \frac{1}{\alpha-1}$, and $E(\mathcal{J}) = \int_0^{M(\mathcal{J})} u_t dt$. The makespan plus energy of the job set under P-FIRST thus satisfies $H(\mathcal{J}) = E(\mathcal{J}) + M(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J})$, and its performance against the optimal is shown in the following theorem.

Theorem 10 P-First *is $O(\ln^{1-1/\alpha} P)$-competitive with respect to makespan plus energy for any set of batched* (Par-Seq)* *jobs, where P is the total number of processors.*

Proof. Since the makespan plus energy of job set $\mathcal{J}$ scheduled by P-First satisfies $H(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J})$, we will mainly focus on the makespan $M(\mathcal{J})$. We bound separately the time $M'(\mathcal{J})$ when all P processors are utilized and the time $M''(\mathcal{J})$ when less than P processors are utilized. Obviously, we have $M(\mathcal{J}) = M'(\mathcal{J}) + M''(\mathcal{J})$.

According to P-First, the execution rate when all P processors are utilized is given by $\frac{P^{1-1/\alpha}}{(\alpha-1)^{1/\alpha}}$. The total work completed in this case is upper bounded by $\sum_{i=1}^{n} w(J_i)$. Hence, we have $M'(\mathcal{J}) \leq (\alpha-1)^{1/\alpha}\frac{\sum_{i=1}^{n} w(J_i)}{P^{1-1/\alpha}}$. We now bound $M''(\mathcal{J})$ when less than P processors are used, which only occurs while P-First executes sequential phases. Since all jobs are batch released, the number of active jobs monotonically decreases. Let T denote the first time when the number of active jobs drops below P, and let $m = n_T$. Therefore, we have $m < P$. For each of the m active job J_i at time T, let $\bar{l}_i$ denote the remaining span of the job. Rename the jobs such that $\bar{l}_1 \leq \bar{l}_2 \leq \cdots \leq \bar{l}_m$. Since P-First executes the sequential phases of all jobs at the same speed, the sequential phases of the m jobs will complete exactly in the above order. Define $\bar{l}_0 = 0$, then we have $M''(\mathcal{J}) = \sum_{i=1}^{m} \frac{\bar{l}_i - \bar{l}_{i-1}}{\left(\frac{1}{(\alpha-1)(m-i+1)}\right)^{1/\alpha}} = (\alpha-1)^{1/\alpha}\sum_{i=1}^{m}\left((m-i+1)^{1/\alpha} - (m-i)^{1/\alpha}\right)\bar{l}_i$. For convenience, define $c_i = (m-i+1)^{1/\alpha} - (m-i)^{1/\alpha}$ for $1 \leq i \leq m$, and we can get $c_i \leq \frac{1}{(m-i+1)^{1-1/\alpha}}$. Let $R = \sum_{i=1}^{m}\bar{l}_i^{\alpha}$, and subject to this condition and the ordering of $\bar{l}_i$, $\sum_{i=1}^{m} c_i \cdot \bar{l}_i$ is maximized when $\bar{l}_i = R^{1/\alpha} \cdot \frac{c_i^{\frac{1}{\alpha-1}}}{\left(\sum_{i=1}^{m} c_i^{\frac{\alpha}{\alpha-1}}\right)^{1/\alpha}}$. Hence, we have $M''(\mathcal{J}) \leq (\alpha-1)^{1/\alpha}R^{1/\alpha}\left(\sum_{i=1}^{m} c_i^{\frac{\alpha}{\alpha-1}}\right)^{1-1/\alpha} \leq (\alpha-1)^{1/\alpha}R^{1/\alpha}H_m^{1-1/\alpha}$, where $H_m = 1 + 1/2 + \cdots + 1/m$ denotes the m-th harmonic number.

The makespan plus energy of the job set scheduled under P-First thus satisfies $H(\mathcal{J}) \leq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\left(\frac{\sum_{i=1}^{n} w(J_i)}{P^{1-1/\alpha}} + R^{1/\alpha}H_m^{1-1/\alpha}\right)$. Since it is obvious that $\sum_{i=1}^{n} l(J_i)^{\alpha} \geq \sum_{i=1}^{m}\bar{l}_i^{\alpha} = R$, comparing the performance of P-First with that of the optimal in Lemma 9, we have $H(\mathcal{J}) \leq (1 + H_m^{1-1/\alpha}) \cdot H^*(\mathcal{J}) = O(\ln^{1-1/\alpha} P) \cdot H^*(\mathcal{J})$, as $m < P$ and it is well-known that $H_m = O(\ln m)$. □

From the proof of Theorem 10, we can see that the competitive ratio of P-First is dominated by the execution of sequential phases of the (Par-Seq)* jobs. Without knowing the jobs' future work, the best strategy for any online algorithm does seem to execute their sequential phases at the same rate. In the following theorem, we confirm the intuition by proving a matching lower bound for any semi-clairvoyant algorithm using sequential jobs only. This result suggests that P-First is asymptotically optimal with respect to the makespan plus energy.

Theorem 11 *Any semi-clairvoyant algorithm is $\Omega(\ln^{1-1/\alpha} P)$-competitive with respect to makespan plus energy, where P is the total number of processors.*

Proof. Consider a batched set $\mathcal{J}$ of P sequential jobs, where the i-th job has span $l(J_i) = \frac{1}{(P-i+1)^{1/\alpha}}$. Since the number of jobs in this case is the same as the number of processors, any reasonable algorithm will assign one job to one processor. From Lemma 9, the optimal offline algorithm has makespan plus energy $H^*(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}H_P^{1/\alpha}$, where H_P is the P-th harmonic number. We will show that P-FIRST performs no worse than any semi-clairvoyant algorithm A. From the proof of Theorem 10, we can get $H_{\mathrm{PF}}(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^{P} \left((P-i+1)^{1/\alpha} - (P-i)^{1/\alpha}\right) l(J_i) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^{P} \frac{l(J_i)}{\alpha(P-i+1)^{1-1/\alpha}} = \frac{1}{(\alpha-1)^{1-1/\alpha}} H_P$. Comparing the performances of P-FIRST and the optimal proves the theorem.

To show $H_{\mathrm{PF}}(\mathcal{J}) \leq H_{\mathrm{A}}(\mathcal{J})$, we construct schedules from A to P-FIRST in three steps without increasing the total cost. For the schedule produced by A, the adversary always assigns the i-th job to the processor that first completes $\frac{1}{(P-i+1)^{1/\alpha}}$ amount of work with ties broken arbitrarily. For convenience, we let the i-th job assigned to the i-th processor. First, we construct schedule A$'$ from A by executing each job J_i with constant speed s'_i derived by taking the average speed of processor i in A. Based on the convexity of the power function, the completion time of each job remains the same in A$'$ but the energy may be reduced. Thus, we have $H_{\mathrm{A}'}(\mathcal{J}) \leq H_{\mathrm{A}}(\mathcal{J})$. According to the adversarial strategy, the processor speeds in A$'$ satisfy $s'_1 \geq s'_2 \geq \cdots \geq s'_P$. We then construct schedule A$''$ by executing each job J_i with speed s'_P throughout its execution. Since we also have $l(J_1) < l(J_2) < \cdots < l(J_P)$, the makespan in A$''$ is still determined by job J_P and is the same as that in A$'$, but the energy may be reduced by slowing down other jobs. Thus, we have $H_{\mathrm{A}''}(\mathcal{J}) \leq H_{\mathrm{A}'}(\mathcal{J})$. Note that the speeds of all processors are the same in A$''$ now. According to Lemma 8, we can construct schedule B from A$''$ such that it consumes constant total power of $\frac{1}{\alpha-1}$ at any time and $H_{\mathrm{B}}(\mathcal{J}) \leq H_{\mathrm{A}''}(\mathcal{J})$. By observing that B is identical to P-FIRST, the proof is complete. $\square$

5 Discussions

In this paper, we assumed a parallel job model where each phase of a job can only take a linear speedup function. However, the model used in [14, 13, 15, 9] does not restrict the speedup function to be strictly linear; instead, they assumed a more general model, where each phase can have a *sub-linear* and *non-decreasing* speedup. How to devise a similar general model that is compatible with the nonuniform speed scaling policy and the semi-clairvoyant processor allocation policy is an interesting problem to consider. In addition, compared to the result of N-EQUI with respect to the total flow time plus energy, Chan, Edmonds and Pruhs [9] showed that MULTILAPS achieves the same asymptotic result of $O(\log P)$-competitiveness as well as the same lower bound of $\Omega(\log^{1/\alpha} P)$-competitiveness. However, their results assumed a different execution model than ours. It is interesting that N-EQUI and MULTILAPS achieve identical asymptotic results in these scenarios under two different execution models, and it should be useful to further illuminate the relation between the two models and identify more fundamental issues in multiprocessor speed scaling. Moreover, it is also desirable to obtain tighter upper or lower bounds for arbitrarily released jobs under either execution model.

For the objective of makespan plus energy, which is considered for the first time in the literature, we have only studied the performance of semi-clairvoyant algorithms on (PAR-SEQ)* jobs. How to deal with jobs with arbitrary parallelism profile and what is the performance in the non-clairvoyant setting remain interesting problems to consider. In particular, comparing the known performance ratios of semi-clairvoyant and non-clairvoyant algorithms with respect to both objective functions, we conjecture that minimizing makespan plus energy is inherently more difficult than minimizing total flow time plus energy, hence is likely to incur a much larger lower bound in the non-clairvoyant setting. The intuition is that a non-clairvoyant algorithm for makespan plus energy can potentially make mistakes not only in speed assignment, but also in processor allocation for the jobs. The former mistake leads to bad performance since jobs that complete early can in fact be slowed down to save energy, and this contributes to the lower bound of semi-clairvoyant algorithms shown in this paper. The situation may deteriorate further in the non-clairvoyant setting as more energy will be wasted or slower execution rate will result if a wrong number of processors is also allocated to a job.

References

[1] S. Albers. Algorithms for energy saving. *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 173–186, 2009.

[2] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *STACS*, pages 621–633, Marseille, France, 2006.

[3] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701, New York, NY, USA, 2009.

[4] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *SODA*, pages 805–813, New Orleans, LA, USA, 2007.

[5] M. A. Bender and M. O. Rabin. Scheduling Cilk multithreaded computations on processors of different speeds. In *SPAA*, pages 13–21, Bar Harbor, ME, USA, 2000.

[6] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

[7] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, New York, NY, USA, 1998.

[8] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[9] H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 409–420, Freiburg, Germany, 2009.

[10] H.-L. Chan, J. Edmonds, and K. Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, Calgary, Canada, 2009.

[11] Z.-L. Chen. Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research*, 129:135–153, 2004.

[12] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167, Philadelphia, PA, USA, 1996.

[13] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, Atlanta, GA, USA, 1999.

[14] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. In *STOC*, pages 120–129, El Paso, TX, USA, 1997.

[15] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, pages 685–692, New York, NY, USA, 2009.

[16] D. Grunwald, I. Charles B. Morrey, P. Levis, M. Neufeld, and K. I. Farkas. Policies for dynamic clock scheduling. In *OSDI*, pages 6–6, San Diego, CA, USA, 2000.

[17] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, second edition, 1988.

[18] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED*, pages 38–43, Portland, OR, USA, 2007.

[19] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[20] J. M. Jaffe. An analysis of preemptive multiprocessor job scheduling. *Mathematics of Operations Research*, 5(3):415–421, 1980.

[21] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, pages 123–134, Salt Lake City, UT, USA, 2008.

[22] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *ESA*, pages 647–659, Karlsruhe, Germany, 2008.

[23] T. Mudge. Power: A first-class architecture design constraint. *Computer*, 34(4):52–58, 2001.

[24] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *WAOA*, pages 307–319, Mallorca, Balear Islands, Spain, 2005.

[25] J. Robert and N. Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *ESA*, pages 741–753, Eilat, Israel, 2007.

[26] D. B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. In *SODA*, pages 448–454, Austin, Texas, USA, 1993.

[27] H. Sun, Y. Cao, and W.-J. Hsu. Non-clairvoyant speed scaling for batched parallel jobs on multiprocessors. In *CF*, pages 99–108, Ischia, Italy, 2009.

[28] M. A. Trick. Scheduling multiple variable-speed machines. In *IPCO*, pages 485–494, Waterloo, Canada, 1990.

[29] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, pages 13–23, Monterey, CA, USA, 1994.

[30] A. Wierman, L. L. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *IEEE Infocom*, pages 2007–2015, Rio de Janeiro, Brazil, 2009.

[31] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, Milwaukee, WI, USA, 1995.

Appendix A. Proof of Lemma 6

Lemma 6 *For any $P \geq 1$, $\alpha > 1$ and $b > 0$, subject to the condition that $\sum_j^P s_j^\alpha = b$ and $s_1 \geq s_2 \geq \cdots \geq s_P \geq 0$, $\max_{1 \leq h \leq P} \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$ is minimized when $(s_1, s_2, \cdots, s_P)$ satisfy $\frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j} = \frac{(h-1)^{1-1/\alpha}}{\sum_{j=1}^{h-1} s_j}$ for all $h = 2, \cdots, P$.*

Proof. To prove this lemma, we transform the stated problem into a convex optimization problem. We then show that our proposed solution satisfies the KKT condition, which is known to be a sufficient condition for the optimality of convex minimization problems. This then leads to the proof of the lemma. First, by introducing a variable y, the original optimization problem can be transformed into the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & y \\ \text{subject to} \quad & \sum_{j=1}^{P} s_j^\alpha = b \qquad (3) \\ & s_j \geq s_{j+1} \text{ for } j = 1, \cdots, P-1 \\ & y \geq \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j} \text{ for } h = 1, \cdots, P \end{aligned}$$

However, the above minimization problem is not convex because its equality constraint (Equation 3) is not linear. Substituting $z_j = s_j^\alpha$, we transform it into a convex optimization problem as follows.

$$\begin{aligned} \text{minimize} \quad & y \\ \text{subject to} \quad & \sum_{j=1}^{P} z_j = b \qquad (4) \\ & z_{j+1} - z_j \leq 0 \text{ for } j = 1, \cdots, P-1 \qquad (5) \\ & \frac{h^{1-1/\alpha}}{\sum_{j=1}^h z_j^{1/\alpha}} - y \leq 0 \text{ for } h = 1, \cdots, P \qquad (6) \end{aligned}$$

For this minimization problem, the objective function and the only equality constraint (Eq. (4)) are linear, the inequality constraints (Inequalities (5) and Inequalities (6)) are convex. Note that Inequalities (6) are convex because of the fact that $1/f(x)$ is a convex function if $f(x)$ is a positive concave function, and $\sum_{j=1}^h z_j^{1/\alpha}$ is concave since $z_j^{1/\alpha}$ is concave for $\alpha > 1$. We have now transformed our min-max optimization problem into a convex minimization problem. We will prove that our proposed solution $(y^*, z_1^*, \cdots, z_P^*)$, which has the form

$$y^* = \frac{h^{1-1/\alpha}}{\sum_{j=1}^h (z_j^*)^{1/\alpha}} \text{ for } h = 1, \cdots, P, \qquad (7)$$

is the optimal solution by showing that it satisfies the KKT condition. Let $x_j = j^{1-1/\alpha} - (j-1)^{1-1/\alpha}$ for $j = 1, \cdots, P$ and apparently we have $x_j > x_{j+1}$. From Eq. (7) and equality constraint (Eq. (4)), we can get $z_j^* = b \cdot \frac{x_j}{\sum_{i=1}^{P} x_j^{\alpha}}$ and therefore $z_j^* > z_{j+1}^*$ for $j = 1, \cdots, P-1$.

To prove $(y^*, z_1^*, \cdots, z_P^*)$ satisfies the KKT condition, we need to show that it satisfies primal feasibility, dual feasibility, complementary slackness, and stationarity. It is not hard to see that the proposed solution satisfies the primal feasibility at Eq. (4), Inequalities (5) and Inequalities (6). Let us now associate multipliers with constraints:

$$\begin{aligned}
\lambda &: \sum_{j=1}^{P} z_j = b \\
w_j &: z_{j+1} - z_j \leq 0 \text{ for } j = 1, ..., P-1 \\
\mu_h &: \frac{h^{1-1/\alpha}}{\sum_{j=1}^{h} z_j^{1/\alpha}} - y \leq 0 \text{ for } h = 1, ..., P
\end{aligned}$$

Since we have $z_j^* > z_{j+1}^*$ for $j = 1, \cdots, P-1$, to satisfy complementary slackness, we get $w_j = 0$ for $j = 1, \cdots, P-1$. Now we need to show that there exists λ and $\mu_h \geq 0$ such that dual feasibility and stationarity are satisfied. To derive stationarity condition, let us look at the Lagrangian function:

$$L(y, z_j, \lambda, \mu_h) = y + \sum_{h=1}^{P} \mu_h \left(\frac{h^{1-1/\alpha}}{\sum_{j=1}^{h} z_j^{1/\alpha}} - y \right) + \lambda \left(\sum_{j=1}^{P} z_j - b \right).$$

Taking derivative of the Lagrangian function with respect to y and z_j, and substituting $(y^*, z_1^*, ..., z_P^*)$ into it, we get the following set of stationarity conditions:

$$\sum_{h=1}^{P} \mu_h = 1, \tag{8}$$

$$\frac{(y^*)^2}{\alpha (z_j^*)^{1-1/\alpha}} \left(\sum_{h=j}^{P} \frac{\mu_h}{h^{1-1/\alpha}} \right) = \lambda \text{ for } j = 1, \cdots, P. \tag{9}$$

Solving Eq. (9), we have $\mu_h = c_h \cdot \lambda$, where $c_h = \frac{h^{1-1/\alpha}\left(\left(z_h^*\right)^{1-1/\alpha} - \left(z_{h+1}^*\right)^{1-1/\alpha}\right)\alpha}{(y^*)^2}$, for each $h = 1, \cdots, P$, and z_{P+1}^* is defined to be 0. According to the values of $(y^*, z_1^*, \cdots, z_P^*)$, we know that $y^* > 0$, $z_h^* > 0$ and $z_h^* > z_{h+1}^*$. Therefore, we have $c_h > 0$ for $h = 1, ..., P$. Substituting $\mu_h = c_h \cdot \lambda$ into Eq. (8), we get $\lambda = \frac{1}{\sum_{h=1}^{P} c_h} > 0$, which implies that $\mu_h > 0$ for all $h = 1, \cdots, P$. Thus, we have shown that the dual feasibility is satisfied. Moreover, there exists λ and μ_h that make our proposed solution $(y^*, z_1^*, \cdots, z_P^*)$ satisfy the stationarity, hence the KKT condition. Therefore, it is the optimal solution for the convex minimization problem, and the corresponding speed assignment $s_j^* = (z_j^*)^{1/\alpha}$ is optimal for the original optimization problem. $\square$